Software Engineering Institute

Quality-Attribute-Based Economic Valuation of Architectural Patterns

Ipek Ozkaya
Rick Kazman
Mark Klein

May 2007

TECHNICAL REPORT
CMU/SEI-2007-TR-003
ESC-TR-2007-003

Software Architecture Technology Initiative

Carnegie Mellon

Table of Contents

List of Figures

List of Tables

Acknowledgments

The authors thank the following people for their careful review and insightful comments: Ashish Agarwal from the Tepper Business School of Carnegie Mellon University, Jason Woodard from the School of Information Systems at Singapore Management University, and Robert Ferguson, Linda Northrop, and Kurt Wallnau from the Carnegie Mellon® Software Engineering Institute.

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Abstract

Quality attribute requirements are a driving force for software and system architecture design. Architectural patterns can be used to achieve quality attribute requirements. Consequently architectural patterns generate value based on the present and future *utility* of the quality attributes they achieve. This report makes the case that architectural patterns carry economic value in part in the form of real options, providing software architects the right, but not the obligation, to take subsequent design actions. The report shows, via a simple example, how an analysis of the options embodied within architectural patterns allows an architect or manager to make reasoned choices about the future value of design decisions, considering this value along multiple quality attribute dimensions.

1 Introduction

Architects must often make architectural design decisions but are typically unable to evaluate their economic impact. Management is often interested in product-level decisions (such as features and quality) but not in the technical details of how those decisions are achieved. These differing interests can lead to inconsistencies between how executives and managers define and foresee value, and how architects can enable or disable those value propositions through their design decisions. This lack of effective communication results in a weak partnership between architects and executives, resulting in missed opportunities to make informed and technically feasible value-driven design decisions. This information exchange is particularly critical when an organization must plan for architecture evolution in the face of uncertain future business and mission goals.

Since software engineering artifacts exist to serve the business goals of an enterprise, optimizing the value of software systems is a central concern of software engineering [Boehm 2000]. Equipping software architects with the ability to reason about value will provide them with the vocabulary and rationale needed to articulate the value-driven impact of architectural decisions to management. This report is the first in a series of reports where we apply economic-driven principles to software architecture—in particular to the selection of architectural patterns—in order to provide better tools for communication between software architects and executives during decision making.

1.1 ARCHITECTURAL PATTERNS

Architectural patterns are intended to aid in creating architectures that meet quality attribute requirements[1] [Buschmann 1996, Bass 2003]. Such patterns are employed by architects to achieve a desired quality attribute behavior, which, in turn, imparts utility to the architecture. We can analyze those patterns to estimate the future value of a system in the face of uncertainty based on the utility of the quality attributes that can be achieved from the patterns' application and based on a prediction of how much the market will value that level of utility.

An *option*, a financial concept, is the right, but not the obligation, to take an action in the future when there is uncertainty. *Real options* apply this concept to real-life investments such as manufacturing plants, information technology investments, and product research and development [Amram 1999, Hull 2006]. Modularity has been an appealing design strategy for creating future value in the form of real options [Baldwin 2000]. Modularity is a quality of a system that consists of various parts that separate cleanly and fit together well. When the design value of an architecture is expressed in terms of options, that architecture is viewed as a real asset [Baldwin 2000]. Well-structured modular designs support modifiability—the degree to which a software system can accommodate changes. Although assigning value to architectural designs during design based on the modularity they provide is an appealing strategy given the inherent uncertainty of future

[1] Quality attribute requirements are those that have significant influence on the creation of the software architecture of a system, such as performance, security, modifiability, reliability, testability, and usability [Bass 2003].

change, that approach ignores the impact of how multiple quality attributes interact when architectural decisions are made. While a modular design supports modifiability well, it may not support, for example, performance as well.

In this report, we introduce and demonstrate a practical method for quantitatively reasoning about the options created through the application of architectural patterns, following a binomial, real options valuation approach [Hull 2006, Amram 1999]. Currently, such decisions are made by architects and project managers in an ad hoc fashion—they simply do not have the tools needed to consider the value implications of choosing a pattern decision with respect to quality attributes and potentially uncertain future market conditions. To address this shortcoming, we introduce an approach that considers patterns and quality attributes as critical in making value decisions. This approach considers cost, value, and alignment with business goals to determine the option value of an architectural pattern. We illustrate our approach by working through a model problem.

A central idea in our work is that an architectural decision, such as the application of a pattern, is analogous to a *financial derivative*. In financial markets, a derivative is a financial instrument whose value depends on, or derives from, the values of basic underlying assets [Hull 2006]. For example, a stock option is a derivative whose value is dependent on the price of the stock. The stock is the underlying asset for the option. We will show how quality attributes and their expected utility in the face of uncertainty act as underlying assets for the valuation of design decisions, similar to the valuation of financial derivatives.

Applying real options theory to the economic analysis of software architecture decisions is not unique to our approach [Baldwin 2000, Sullivan 1999, Erdogmus 2002b]. Designers are typically trained to reason about patterns and to some extent quality attributes but typically are not trained in using real options theory for software design. Software architects clearly recognize that quality attributes derive primarily from a system's software architecture, and consequently quality attribute requirements are a driving force for architectural design [Bass 2003]. Using quality attribute requirements as the main driver for generating and assessing value is the significant difference of our approach, and we think it provides important insights. We also think it's realistic: in the real world, no architect can afford to consider a single quality attribute in isolation, and yet the economic tools for considering multiple quality attributes don't yet exist. Our approach provides architects with specific guidance for selecting design patterns in light of utility and uncertainty of architecturally significant requirements and valuing an overall architecture—all of which leads, in the end, to better decision making.

1.2 ABOUT THIS REPORT

1.2.1 Objectives

In this report, we draw on real options theory [Amram 1999, Merton 1998] to analyze the value of architectural patterns in terms of the quality attributes they can help software architecture achieve. Our goal is to provide guidance for making economic design tradeoffs that are attuned to business goals in which architectural flexibility and evolvability are desired. This report addresses these questions:

- How can architectural patterns be evaluated for their real option values?

- Based on such an evaluation, can suggestions about *when* to apply which pattern be generated in order to support architectural evolution?
- Can the valuation of architectural patterns reveal key insights for architectural decision making with respect to quality attributes and business goals?

This report contributes to earlier efforts in applying real options analysis to design valuation by investigating the valuation of architectural patterns. It demonstrates how to value design choices at a level of abstraction that is appropriate for architecting using a binomial option-valuation model as a technique for managing and preparing for architectural evolution.

1.2.2 Organization

This technical report is organized as follows. Section 2 introduces key concepts in real options theory and analysis, along with a discussion of the challenges of applying the theory to valuing software architecture. Section 3 presents our model problem. We summarize related work and provide conclusions along with pointers to future work in Sections 4 and 5, respectively.

2 The Theory of Real Options

2.1 KEY CONCEPTS

An option is the right, but not the obligation, to take an action in the future. *Real* options are those that affect nonfinancial assets. While real options theory has attracted enthusiasm, research in this area shows that rigorous, real-world application of the ideas has been slow, perhaps because the research and application focus has been on the theory's technical aspects, rather than on the strategic way of thinking it introduces [Amram 1999]. Options are especially valuable when there is uncertainty due to a lack of information. The theory is typically applied in such a situation because managing the uncertainty requires flexibility in investment decisions (which are often contingent on initial investments) when there are multiple possibilities for future growth and mid-course strategy corrections are inevitable [Amram 1999]. Waiting can be valuable when it allows investments to be postponed until the uncertainty is resolved.

The components of real options analysis include

- the decision to be made
- a characterization of the uncertainty
- the decision rule

The decision rule is a simple mathematical expression that indicates when the decision should be made and helps identify the critical parameters that architects need to observe during the decision-making process.

2.2 ARCHITECTING AS AN ECONOMIC ACTIVITY

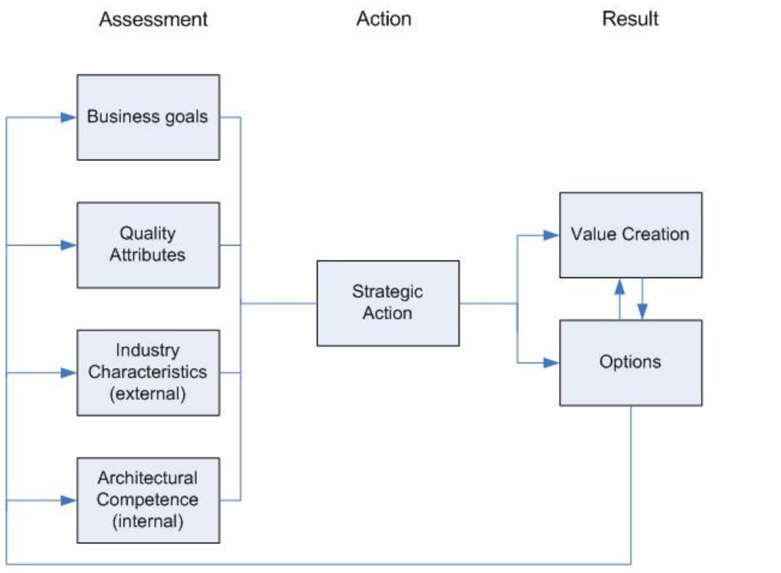
Clearly, the formulation of real options is applicable to architectural design decisions. Investments are frequently contingent on prevailing and future market conditions (e.g., the inclusion of a third tier in a three-tier architecture makes it easier to subsequently modify business logic). Also, the large degree of uncertainty that is characteristic of the future makes it advisable to hold off on making decisions until you have more information (e.g., if the system is successful, input events might increase by two or more orders of magnitude). Applying real options theory to the economic analysis of software architecture decisions has been addressed previously [Baldwin 2000, Sullivan 1999, Erdogmus 2002b]. However, our treatment of using quality attribute requirements as the main source for calculating value provides new insights. In particular, our approach guides the selection of design patterns, the elicitation of architecturally significant requirements, and the overall valuation of designs.

Managing options in a project requires the ability to reverse strategic decisions as more information becomes available. For software architecture, these decisions need to weigh several aspects of the design problem such as business goals (e.g., shorten time to market, increase market share), resources (e.g., access to the information needed to make the decision, developer time), and key quality attributes (e.g., search latency should be less than 1 second, system should be 99.999%

available). However, tradeoffs often need to be made among and within these dimensions. For example, given a problem where modifiability and performance are architecturally significant concerns, how modifiable should the architecture be? What level of performance is desired? Is it better for the architecture to be highly modifiable, while possibly compromising performance or to have high performance, while possibly compromising modifiability? Understanding quality attribute utilities—the value that is derived from an architecture when it achieves a particular level of a quality attribute—can help architects structure and revise strategic architecting decisions made using real options. Such an understanding helps inform the architects about the value of making tradeoffs.

For example, in the case above, what is the value of making the architecture more modifiable or perform better? If the system's modifiability is poor and new features take a long time to implement, the utility of increased modifiability might be substantially higher. If systems based on this architecture already have adequate performance, the marginal utility of performance improvements may be low. But even this information is insufficient to make a truly informed architectural decision. The key to such a decision is to understand the *potential future value* of increased modifiability or performance, the *costs* of achieving that potential value (in terms of architectural strategies pursued today), and the *likelihood* that such an increase in modifiability or performance will be needed.

A design process informed by real options thinking is a strategy-creation process. In architecting, this way of thinking corresponds to staging architectural decisions and deciding what needs to be done and when, based not only on requirements but also in light of changing business goals and new information as it becomes available. The real options approach identifies and values opportunities through the management of uncertainty. The payoff at one stage of development includes the option to go onto the next. Economics-driven architectural thinking not only requires more up-front, strategic work, but it also forces the architect to know more about the design and requires direct involvement of business staff. This knowledge depends on the ability to identify future risk exposures as a function of business goals and on the management of quality attributes, based on the organization's ability. Those dependencies are outlined in Figure 1.



Adapted from Credit Suisse First Boston's Strategy Formulation and Real Options Model into Architecting [Mauboussin 1999]

Figure 1: Framework for Options Analysis in Software Architecting

2.3 VALUATION MODELS

We examine three option valuation models:

- Black-Scholes-Merton, which involved work originally done by Black-Scholes [Black 1972] and later extended by Merton [Merton 1998]
- binomial analysis [Amram 1999, Hull 2006, Cox 1979]
- Baldwin and Clark's more recent Net Option Valuation (NOV) model [Baldwin 2000]

Black-Scholes-Merton and binomial models both emerged from the valuation of financial options. Baldwin and Clark's model, on the other hand, looks closely into options created within designs, particularly how to calculate the option value of modularity using design structure matrices (DSMs). The applicability of these models to software architecting is appealing because they offer a method for estimating the cost and benefit of design decisions in the face of uncertainty that is better than that provided by analyses such as return on investment or discounted cash-flow analysis [Baldwin 2000, Favaro 1998].

2.3.1 Black-Scholes-Merton

Fischer Black and Myron Scholes' model mimicked the behavior of an options portfolio via a *tracking portfolio* (also referred to as a *replicating portfolio* or *twin security*). It has the same risk characteristics as the original project for which an option exists and demonstrates a similar fluctuation in value. Robert Merton added to the model the concept that option prices are determined relative to other prices in the market so as to preclude any arbitrage opportunities. *Arbitrage* is the simultaneous buying and selling of one security at two different prices in two different markets, which results in profits without risk. The Black-Scholes-Merton valuation model is well suited for options that have a single expiration date (the date by which a decision must be made—this is known as a "European" option) and a single source of volatility. However, some have argued that this model does not scale very well when staged decisions are needed [Amram 1999, Favaro 1998]. In software architecture, such decisions about design options are often needed. Some authors have also noted that design options do not have underlying assets to replicate or trade, as required by the Black-Scholes-Merton model [Baldwin 2000]. Bahsoon and Emmerich use the Black-Sholes-Merton model for valuing refactoring in software. In their application of the model, they make the assumption that there is a fixed expiration date for decision making [Bahsoon 2003].

The software architecting process occurs over a period of time up until a deadline (perhaps based on a software release date), which is similar to the expiration date of a financial stock option. Analytical methods such as Black-Scholes-Merton are not suitable for approximating the changes that may occur over time for valuing options that can be exercised at any time up to an expiration date (an "American" option). Numerical methods that look at discrete intervals, such as event trees, are preferred when there are multiple sources of uncertainty and multiple decision points [Favaro 1998, Erdogmus 2002a].

2.3.2 Binomial Option Valuation

A binomial tree is an event tree that represents different possible paths that might be followed over the life of an option. A binomial tree depicts two outcomes at each decision point: a favorable outcome in the uncertain variable (an upward move in value) or an unfavorable outcome (a downward move in value). The Black-Scholes-Merton formula assumes constant market volatility. The discrete binomial approach, on the other hand, allows for modeling changes in volatility at each time period [Amram 1999]. Moreover, since the formula does not assume a fixed time period, it is possible to create sensitivity analyses for different intervals of time until the expiration of the option.

In a binomial tree, the value of the option at final nodes is the same as for a European option, where the option can be exercised only at the expiration date. The nature of the timing of software design decisions acts analogous to an American option, where the option can be exercised any time until a given expiration date. In valuing an American option with a binomial tree, the procedure is to work the tree from the end to the beginning, testing at each decision point whether exercising the decision rule before the expiration of the option is optimal. The value of the option in these nodes is the maximum value of the option and the payoff from exercising the option.

Typically in the real world, architecture decisions are not made at a single predetermined deadline. Therefore a binomial approach, where the exercising window can be broken down into several intervals, appears to be a good fit in terms of the exercise time of the option. Erdogmus and Favaro use binomial trees in their economic analysis of how extreme programming design principles create options (where it is considered valuable to defer decisions when requirements are highly uncertain and volatile) [Erdogmus 2002b]. Binomial-tree-based valuation has also been investigated in relation to software reuse [Favaro 1998] and prototyping [Sullivan 1999].

2.3.3 Net Option Value of Modularity in Design

Baldwin and Clark's Net Option Value (NOV) model [Baldwin 2000] is based on their recognition that modularity adds value by creating design options because it allows designers to independently investigate and replace designs. The dependencies between modules are modeled using a DSM. Their model addresses design options, identifying design as a value-seeking process.

Baldwin and Clark characterize the computer industry as collectively engaging in a process of examining the different options that modularity creates. These options often emerge unplanned. Knowing these industry forces, architects can embed options in their designs to support more effective and efficient evolution of architectures. They identify six modular operations: splitting, substitution, augmenting, exclusion, inversion, and porting. The design rules—parameters used as interfaces between modules that are not likely to change—guide how the operations are used. NOV valuation is based on the value of independent experiments that can be conducted on each module, resulting in the selection of the best-fitting design, hence increasing the total option value of a modular design. The NOV model treats the value of a modular design as a portfolio of options. Modularity creates options based on the opportunities it creates in applying the operators, hence creating alternative designs. The six operators proposed are a reasonable fit with the process of architecting. For example, the Attribute-Driven Design (ADD) method (developed by the Carnegie Mellon® Software Engineering Institute [SEI]) focuses on decomposing the system based on quality attributes and allocating functionality recursively [Bass 2003, Wojcik 2006]. The decomposition process implicitly involves using the six modular operators based on the tactics, patterns, and quality attributes associated with the system and the expertise of the designer.

The NOV model formulates the total option value of a system as the summation of each module's value, which in turn is a function of volatility, denoted as technical potential. This measure of technical potential looks at the standard deviation of the outcomes possible in a module, whereas the volatility in the other models is a historical market-based value. The NOV model has been used in valuing modularity in object- and aspect-oriented designs [Sullivan 2001, Lopes 2005, Sullivan 2005]. The work of Sullivan and colleagues extends the DSM and NOV model by including environmental variables to calculate the value of a design [Sullivan 2001, Cai 2006].

Architects need to understand their designs in terms of how they address different quality attributes to make sound design decisions [Bass 2003]. Modularity becomes an appealing design strategy when the need for easy modifiability coexists with uncertainty of change. However, architecting as a series of module decisions does not accurately abstract the architecting process or consider the potential resulting structural impact of quality attributes on the architecture that may be better abstracted with patterns and tactics [Bass 2003]. For example, modularity to achieve modifiability does not suffice to make sound architectural decisions in addressing usability. Usability is often captured in the architectural level as a series of decisions encapsulated within commonly known modular patterns, such as Model-View-Controller. However, many architecturally significant usability decisions require pattern decisions other than modularity [Golden 2005]. The NOV model values modules from the perspective of design dependencies and flexibility, but the current formulation provides no insights about how quality attributes other than modifiability may affect the valuation.

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

2.4 THEORETICAL CHALLENGES

Valuation of real assets in terms of options takes advantage of the models presented in Section 2.3. Several challenges have been reported that potentially affect the applicability of the models in the context of design. We group the challenges that apply to software architecting as follows. We also note how we deal with these challenges in our formulation.

1. **estimation of volatility:** Volatility is used as a measure of price/value fluctuation, which may also be the key uncertainty input. Volatility is challenging because, depending on the context of the problem, it may refer to an historic or implied value [Hull 2006]. For example, in the context of a stock option, volatility may refer to an *historic* value, as a measure of past fluctuations in the stock price. That is an observed value and can be calculated as the standard deviation of the log of price returns. Volatility can be calculated by using a current option price and solving for volatility as output; this is referred to as *implied* volatility. To calculate a fair value, an options model requires forecasting the volatility of the price over the option's period to expiry. This value can be calculated based on historic or implied volatility values.

   For the purposes of software architecting, in addition to the challenges of picking a model to forecast volatility, several other bottlenecks exist. Several attributes may be observed in software for volatility, similar to how a stock's price is observed. Requirements changes, market response, or the changing price of an outsourced key design component may all serve as parameters by which to observe volatility. Often, historical design data to chart volatility does not exist.

   estimation of volatility in the model problem: We assume volatility to predict the uncertainty of favorable and unfavorable outcomes that could be elicited from the stakeholders, similar to how it is done in an evaluation using the SEI Cost Benefit Analysis Method (CBAM) [Asundi 2001b].

2. **price of an option as a function of an underlying replicating portfolio:** Options may be valuable, but they incur an up-front cost to acquire. The ability to realize this value depends on how well uncertainty is managed and how well opportunities (that arise as more information becomes available) are evaluated and acted on. For financial stocks, this up-front cost to obtain the option is calculated by creating a replicating portfolio and ensuring that there are no arbitrage opportunities in the market—that is, benefits cannot be accumulated due to a price discrepancy in the price of the option. These are the key ideas that led to the creation of the Black-Sholes-Merton model. The price of an option is calculated as a fair cost to pay to obtain the option in a risk-free market without any arbitrage opportunities. In real options, the underlying asset is not typically traded in the market, and its price is not observable. While there are efforts to incorporate non-tradeability and non-observability into the models to make them suitable for real options as well [Merton 1998], the different nature of the underlying assets in real options is often listed as a critical bottleneck in assuming correctness from the models.

   formulation of underlying replicating portfolio in the model problem: For many situations, the starting point is an existing system. Every system embeds a set of options. However, as we consider evolving an existing system, we are often less concerned with the option value of the system as it exists (we have already spent the resources to get to this point) than we are with how future design alternatives create *new* options. In this case, we want to calculate the *change* in option value associated with the change in the design as opposed to the absolute option value. This is the approach we use in this report. We also assume that the existing system and a risk-free interest rate (that could be obtained from the market) act as sufficient underlying assets for our purposes in the model problem.

3. **exercise strategy:** A financial option can typically be exercised only once, either at a given deadline if European or any time up until a predetermined time if American. When talking about real options in design projects, such as software architecting, a design strategy that provides some design flexibility may actually permit options being exercised multiple times. Modifiability is an example of such a strategy: it allows for multiple additions and, hence by definition, multiple equally fruitful opportunities for exercising the option. The experimenting concept introduced by Baldwin and Clark also is an example; by conducting multiple experiments on multiple models, a designer creates multiple exercise opportunities [Baldwin 2000]. The multiple-exercise options within American options exist in energy derivatives and interest-rate-based financial products as well. One approach followed in valuing such multiple-exercise options is the Monte Carlo simulation method [Meinshausen 2004]. The nature of design projects that permit multiple exercise opportunities is a clear divergence from the existing option-valuation models; the applicability of Monte Carlo simulation methods is an area of active research.

   formulation of multiple-exercise options in the model problem: As our problem space spans a short period of time, we generate the value of our multiple-exercise options by independently taking into consideration each exercise strategy and adding the resulting option values. Doing so allows us to incorporate into our model problem options, such as modifiability, that create multiple-option exercise opportunities once embedded into an architecture.

3 City Information System Example

Real options analysis is an applicable valuation technique when the following conditions exist:

- **uncertainty**: There is uncertainty about what may happen in the future.
- **business goal**: The uncertainty is important for managing or achieving a business goal.
- **new information**: The business should be in a position to exploit new information when it becomes available. The assumption is that new information reduces uncertainty and opens up different opportunities.
- **action today**: Recall that an option is defined as the right, but not the obligation to make a decision in the future. To obtain this right (i.e., to hold the option), a stakeholder must take some deliberate value-driven action today.
  - **possibility of future design choices**: The action taken today opens up some future design choices that may not be as feasible today.
  - **possibility of future value**: The future choices create opportunities of value.

3.1 THE CONTEXT

As mentioned, the value of software design decisions can be modeled as an American option, where the option can be exercised any time until a given expiration date. In valuing an American option with the binomial model, the procedure is to analyze the tree from the end to the beginning, testing at each interval whether early exercise of the option is optimal. The value of the option in these nodes is the maximum of the option's value and the payoff from early exercise.

The model problem that we will use as a running example is a city information system (CIS) Web site, provided by BizCo. This is an application that operates in an inherently distributed environment. BizCo makes its profit based on the number of hits its Web site gets. Users retrieve information using a Web browser. *Information resources* host data about city events, places, and the like. BizCo's management believes that more information resources carried in the system will attract more users.

BizCo's current system uses the simple client-server architecture shown in Figure 2. In order to access an information resource in this system, its location must be hard-coded and the service must be executed in the server. While this design is simple and has acceptable latency, it makes the addition of new resources cumbersome. When the information provider receives a request from a client, it runs the appropriate service and returns the information. The system is expected to evolve over time through the addition of new information resources. That addition may also result in an increase in the number of users, which in turn may cause the availability of the system to become a critical quality concern.

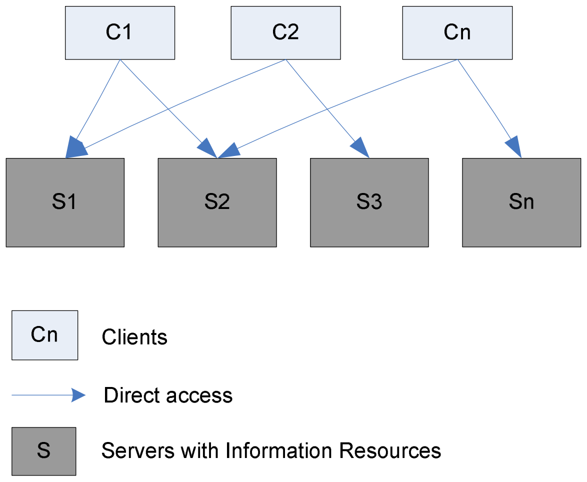


Figure 2: BizCo's Current System Architecture

Our goal is to demonstrate that patterns carry option value based on the quality attributes that can be achieved with them. We are interested in detecting how the design value changes as the quality attribute requirements change over time, such as the need to add new resources, be more available, or handle more users. Guidance for architectural patterns is often described in the context of greenfield projects where development occurs from scratch or in cases where future requirements are understood and may be planned for. However, more frequently, software is not developed from scratch as it has become part of an organization's infrastructure: wholesale replacement is rarely an option, and future changes are frequently not well understood. Nevertheless, it is still necessary to evolve existing software to serve emerging markets, new needs, and new business opportunities, such as including different cities' information. This task is even more challenging when the new conditions are uncertain.

We use a binomial option analysis approach to model this problem because it allows us to observe the changes in design value at each time interval as a result of changes in quality attributes. We follow the binomial tree calculation technique introduced by Cox and colleagues [Cox 1979], Amram and Kulatilaka [Amram 1999], and Hull [Hull 2006]. Our approach focuses on making valuation estimates based on the quality attribute utilities to be gained from the application of architectural patterns.

In the next section, we will introduce our CIS model problem with its evolution and uncertainty concerns. We will present the motivating quality attribute requirements, their expected utilities, and the patterns we consider for addressing these attributes. We then present the valuation calculations and reasoning. We conclude by summarizing the valuation.

3.2 CIS AS A REAL OPTIONS PROBLEM

We make a set of assumptions to structure the CIS problem for valuation via real options analysis.

- **uncertainty:** The ability to find new resources willing to contribute information is a source of uncertainty. BizCo's marketing department predicts that more information resources will bring more users, but it cannot accurately predict the demand—another source of uncertainty. More users will mean that availability requirements will become more stringent. Currently, the system has an availability goal of no more than five hours of downtime per day.

- **business goal:** BizCo would like to increase its market share to 10% of the CIS market. Doing so requires increasing the number of new information resources, which is predicted to increase the number of users.
- **new information:** BizCo is looking into several new collaborations for providing a more diverse information pool. For the sake of simplicity, we assume that any new information resources become available monthly.
- **action today:** BizCo is seeking answers to the following questions: How and when should it evolve the current architecture in response to new user and information-source demands? The action to be taken today involves two considerations:
  - **possibility of future design choices:** BizCo would like to know how to stage architectural decisions and determine which ones bring more value relative to its business goals.
  - **possibility of future value:** If BizCo sticks with the existing client-server architecture, management believes that the company will not be able to respond to future demands, such as including different cities' information. Hence, it may lose market share.

The determination of the action to be taken today in this interpretation of the CIS problem can be conducted via real options analysis, which we demonstrate next.

3.3 UNCERTAINTY

We characterize uncertainty in the context of key quality attributes. Given this statement of the CIS problem, BizCo faces uncertainty in terms of how much modifiability and availability to plan for, and invest in, in the architecture.

BizCo's stakeholders also make the following information available:

- The existing client-server design is costly to maintain, despite its performance advantages due to direct client access.
- The company is uncertain about the number of information resources that will need to be added each month and therefore cannot accurately evaluate the future value of moving to an easier-to-modify architecture today.
- The company does not want to sacrifice performance at the expense of modifiability.
- As the number of users increase, availability requirements are expected to become more stringent.

Uncertainty in modifiability: The uncertainty for modifiability is the number of new information resources that will need to be added. If a new information resource needs to be added every month, with the current architecture, BizCo does not expect to be able to incorporate it into the system in a timely fashion. Hence, the product would lose market share.

Uncertainty in availability: The uncertainty for availability is how critical downtime will be for users. Currently, the system is down from 3 AM – 8 AM for regular maintenance. When new information resources become available, for example from different time zones, five hours will not be acceptable downtime and may need to be reduced to a maximum of one hour per day, or even less.

3.4 DESIGN CHOICES

The architects have identified the possible tactics and patterns shown in Table 1 and would like to evaluate their design value. The notion of a tactic, defined as "a design decision that influences the control of a quality attribute response," guides the selection of appropriate architectural patterns and frequently tailors them [Bass 2003]. For example, information hiding is a tactic that is commonly resorted to when changes are anticipated. Patterns that modularize and isolate the characteristics that are most vulnerable to change implement that tactic.

Table 1: Tactics Identified to Address Quality Attribute Concerns

Quality Attribute	**Tactics**	**Patterns to achieve tactic**
Modifiability	Prevention of ripple effects: hide information, use an intermediary	Client-Server, Proxy, Broker, Client-Dispatcher-Server
Performance	Resource demand: reduce computational overhead Resource management: maintain multiple copies	Proxy, Shared Memory
Availability	Fault detection: ping/echo Fault recovery: active redundancy	Client-Server, Proxy, Broker, Client-Dispatcher-Server,

The commonly used technique for meeting volatile modifiability requirements is to modularize the system so that there is a decoupling of information dependencies. In this example, we consider two candidate tactics to achieve this decoupling: leveraging information hiding to isolate and protect changes within one module by means of interfaces and using an intermediary to translate information from one form to another to manage change propagation.

BizCo's lead architect is considering using the following candidate patterns in order to realize these tactics: the current Client-Server approach, Client-Dispatcher-Server, Proxy, and Broker [Buschmann 1996, Bass 2003]. Evaluating which of these patterns makes more economic sense requires planning for not only the strengths of these patterns but also their weaknesses. For example, while the Broker pattern is well suited for a problem where modifiability is required, its slower performance and low fault tolerance may eliminate its use. Similarly, while a dispatcher component in the Client-Dispatcher-Server pattern creates flexibility by easing the exchangeability of servers, the dispatcher component itself is vulnerable to changes in its interface. In Table 2, we summarize the technical aspects of these patterns, based on the work of Buschmann and colleagues [Buschmann 1996].

Table 2: Broker, Proxy, and Client-Dispatcher-Server Patterns

	Benefits	Liabilities
BROKER	Clients and servers do not need to know each others' locations, since all requests are handled through the broker.	Slower runtime performance
	Changeability and extensibility of components	Low fault tolerance
	Reusability of existing services	
PROXY	Decoupling of clients from location of servers (with better runtime in exchange for some loss of flexibility)	Efficiency: slower due to indirection
	Efficiency at lower cost by using "load on demand" strategy	Loading on demand does not work when originals are highly dynamic
CLIENT-DISPATCHER-SERVER	Exchangeability of servers	Efficiency: slower due to indirection and explicit connection establishment
	Allows for deferring decisions and preparing for later conversion to a distributed system	Dispatcher is a central component, and changes to the interface of this component creates modifiability vulnerabilities.
	Fault tolerance: new servers can be activated without impact	
CLIENT-SERVER	Enables encapsulation of the handling of shared resources on the behalf of multiple clients with a simple structure	Clients need to explicitly know the location of the servers to access services, which hinders scalability and modifiability when new servers need to be added.
	Enables processes that reside on the servers to be modified independent of the clients	Slower runtime performance due to communication between clients and servers

3.5 KEY DEFINITIONS

Several pieces of information need to be considered in order to construct a meaningful economics-based valuation: the current state of the architecture, market expectations, architectural patterns and their behavior with respect to quality attribute responses, the cost of making the changes, and the cost of ongoing maintenance. The elicitation of such information from project stakeholders is a feasible and commonly applied technique [Bass 2003, Dobrica 2002]. Table 3 summarizes the response measure estimates for each of the candidate patterns, with respect to each of BizCo's driving quality attribute scenarios. Making such estimates (and predictions) is a normal duty of a software architect and may be done via experience, building prototypes, or the creation of an analytic model.

Table 3: Quality Attribute Responses

	Modifiability	**Performance**	**Availability**
	person-day	seconds	hr downtime
Client-Server	7	0.5	5
Client-Dispatcher-Server	2	3	1
Proxy	2	2	1
Broker	1	5	3

A financial (stock) option is the right, but not the obligation to buy (or sell) some amount of stock at some point (or interval of time) in the future. Determining the value of a stock option is based in part on the value of the stock [Hull 2006]. In real options analysis, the value of the project's expected cash flow is similar to the *stock price* in a financial option. We use a different approach. We associate *utility* with different aspects of the system such as levels of functionality and quality. Total system utility is our stock price equivalent. The cost of making the change enabled by the option (such as adding a new component) is equivalent to the *exercise price*, and the time until the opportunity disappears is the *time to expiration*. *Volatility* represents project value fluctuation, and finally the *risk-free interest rate* is assumed to be a known market value [Hull 2006, Amram 1999, Favaro 1998] (or possibly a known organization-specific value).

We map these components to our analysis as follows. The *decision* BizCo needs to make is which design pattern to embed in the architecture and when. *This decision is evaluated based on the value of the options created by the selected pattern.* The cost of switching to a new architecture pattern is similar to the premium to be paid in buying a stock option—that is, the cost of the option, which we refer to as the *switching cost*. The *exercise price* is the maintenance cost that needs to be spent in each alternative for adding new information resources. The time interval *t* in our example is three months.[2] The risk-free interest rate *r* is 6% per year or 0.5% per month.

In the CIS example, we use total, system-wide utility value instead of stock price. The system value is determined by summing several dimensions of utility obtained from the quality attributes the system achieves.

$$S = V_S + v_{IP} + v_A \qquad (1)$$

where

- S is total system value.
- v_{IP} is the utility value from modifying the system with information providers.
- v_A is the utility value from having more availability.
- V_s denotes the rest of the current system's value prior to making any changes.
- V_{IP} is the expected outcome associated with modifiability.
- V_A is the expected outcome associated with availability.

[2] Company planning cycles, for example, are often based on quarterly intervals. The example uses monthly intervals for demonstration purposes. The time intervals—hence the risk-free interest rate to be used—may be adjusted depending on the availability of information and the specific industry in which a company operates.

The uncertain quality attributes can be elicited from the key stakeholders by means of an evaluation exercise, using, for example, the SEI Architecture Tradeoff Analysis Method® (ATAM ®) [Clements 2002]. Figure 3 shows the expected utility of different numbers of information resources.

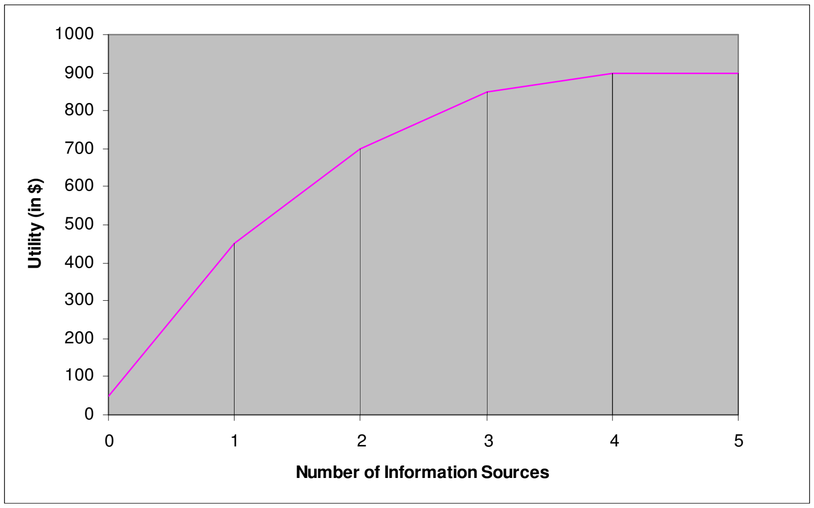


Figure 3: Utility Versus New Information Resources in Time

Figure 4 shows the predicted favorable and unfavorable outcomes at each time period using a binomial tree. For example, in Figure 4, at cell G, after three months, $850 is the utility associated with three information providers. This value is assumed to be elicited from the stakeholders and charted as a utility graph, as represented in Figure 3. The associated graph in each figure charts utility as a function of each uncertain variable. (A more accurate depiction of the utility curve, would be utility surface, where utility is a function of time and the uncertain variable. We are actually plotting a line on that surface.) The explanations at the leaf nodes describe the state observed at the end of the given three-month period with the paths reaching to that particular node. For simplicity, we graph utility in terms of dollars.

® Architecture Tradeoff Analysis Method and ATAM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

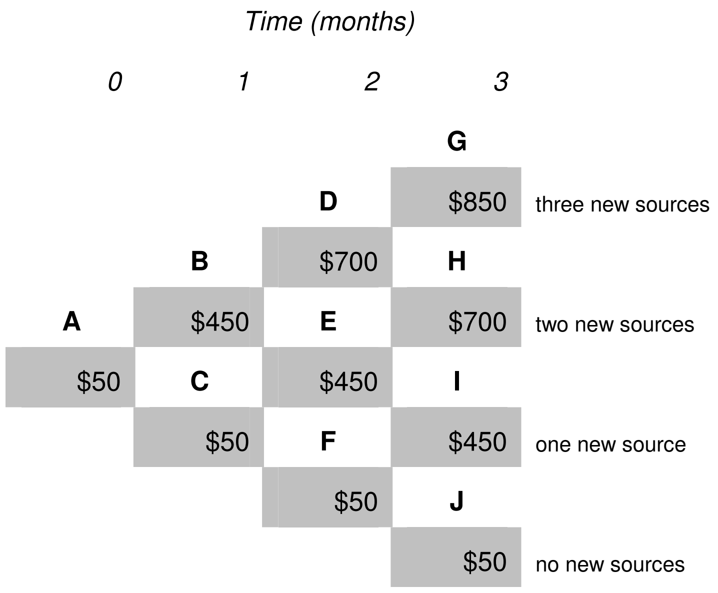


Figure 4: Modifiability – Uncertainty in the Number of New Information Resources

The outcomes of the uncertainty variable in availability—measured as downtime in hours—have an impact on the utility. There may be several different outcomes. For example, investing more in availability and making the operation a 24x7 one may create no utility, which would bring no extra value. Or perhaps the more available the system is, the better it will be. Or after an initial increase in utility, the system may reach a point where investing more in availability does not create more value. This last outcome is shown in Figure 5, where investing more in availability beyond six hours of downtime does not create any significant added value. The exercise price in the case of availability is the maintenance cost related to adding new information resources.

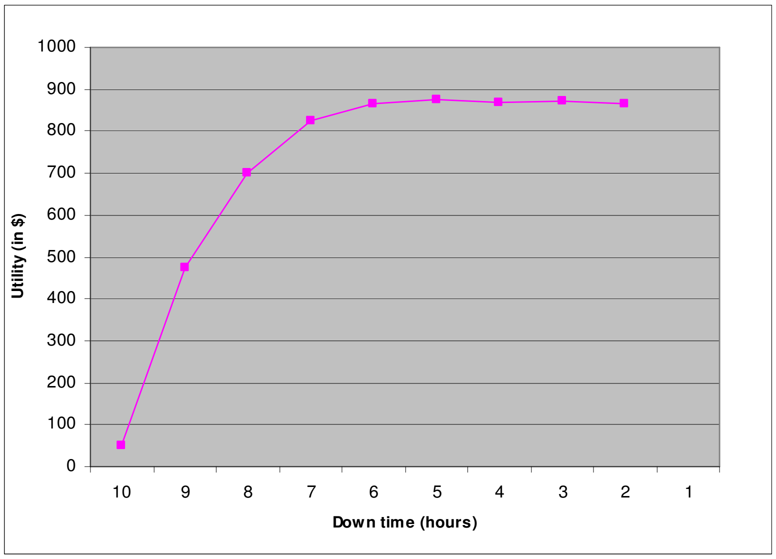


Figure 5: Utility Versus Downtime Due to Adding New Information Resources

Figure 6 shows the predicted favorable and unfavorable impact of utility to the total system value depending on how critical availability turns out to be for BizCo.

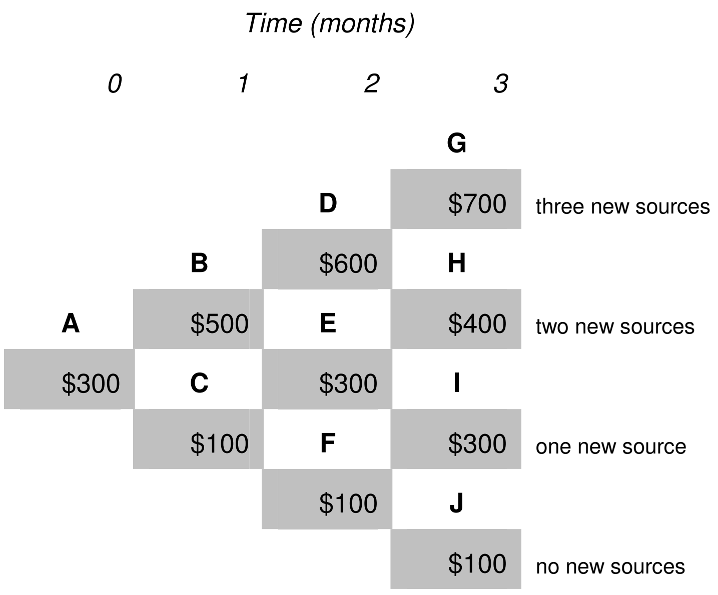


Figure 6: Availability – Uncertainty in the Importance to Users

One approach to real options calculation bases the prediction of favorable and unfavorable outcomes by calculating coefficients based on the volatility of the expected outcome [Amram 1999, Hull 2006]. Our starting point is assumed values for these potential favorable and unfavorable outcomes at each level of the binomial tree. In an ideal setting, these predictions would be based on volatility empirically observed in real design situations [Baldwin 2000, Sullivan 2001]. Such a knowledge base could be built over time, with the accrued experience of repeatedly building similar kinds of systems. In this paper, we are not concerned with how to predict the market's reaction to various system changes but rather how to valuate the switching cost and the most feasible architectural pattern choices, given such a prediction.

The utility values should be thought of as contributors to the system's value. For example, for modifiability, the utilities reflect how the market would "feel" about the system if it had various numbers of information providers. As depicted in Figure 3, more information resources provide more utility. For availability, the utilities reflect how the market would "feel" about various levels of downtime for routine system maintenance. We map the possible quality attribute situations, such as the number of new information resources, on a binomial tree and plot the corresponding utility of each outcome during the three-month period of our analysis.

In addition, we use the following notation:

- C_d, C_p, C_b : switching costs (that is, option prices) for introducing a dispatcher, proxy, or broker, respectively
- C_{dm}, $C_{pm,}$ C_{csm} : maintenance costs (that is, exercise prices) for modifying the dispatcher, proxy, or broker with a new information server, respectively. We generalize this variable as C_{xm}.

The switching and maintenance costs we assume are shown in Table 4. These cost assumptions are conjectural but consistent with the analysis of the patterns presented in Table 2. The switching cost is a one-time expense required for making the architectural changes, similar to the premium paid to obtain a stock option. Note that this represents the *actual cost* of acquiring the option in this example. We will use the theory of binomial option valuation to determine if this is an appropriate price. The maintenance cost of modifiability is incurred *every time* an information resource is added. The maintenance cost of availability is incurred as a result of dealing with downtime as new information resources are added.

Table 4: Cost Assumptions

	Switching cost	Maintenance cost (modifiability)	Maintenance cost (availability)
Client-Server	0	3000	550
Client-Dispatcher-Server	4100	1000	100
Proxy	4100	1000	100
Broker	4500	320	500

3.6 VALUATION OF CIS WITH REAL OPTIONS ANALYSIS

We need to calculate the value of each outcome at each step of the tree in order to understand the value of the options we may have within the CIS problem. Our approach is based on the binomial options pricing model. For a two-step binomial tree, consisting of the starting point and the states one level further in time, the option price formula is

$$f = (pf_u + (1-p)f_d) / (1+r) \qquad (2)$$

where

- f is the option price.
- Subscript *u* indicates that the value of the system (in our case, the stock equivalent) goes up.
- Subscript *d* indicates that the value of the system goes down.
- *uu* in a multi-period timeline would refer to two subsequent favorable outcomes.
- *ud* in a multi-period timeline would refer to one favorable and one unfavorable outcome and so on.
- r is the risk-free rate of return.
- p is a coefficient as defined in Equation 3.

$$p = (1+r - d) / u-d \qquad (3)$$

Here, u is the factor by which the stock price increases, and d is the factor by which the stock price decreases. In our case, the u and the d values are calculated based on the predicted system values we elicit from stakeholders. There are more general formulas for n-level binomial trees.

In the CIS example, we apply the current Client-Server design and the Broker, Client-Dispatcher-Server, and Proxy pattern alternatives once for each alternative and then evaluate the results. We assume that the Client-Server has some value denoted by $S_{cs}(0)$, where 0 indicates that the time equals 0 months. We also assume that this value is totally due to the utility associated with providing some number of information providers plus the utility associated with some level of availability at time t. Therefore

$$Scs(t) = VS + vIPcs(t) + vAcs(t) \qquad (4)$$

The same is true when using the Dispatcher, Proxy, and Broker patterns. We can now interpret the above formula:

- S(1) = uS(0). When the system value increases, S(1) is also denoted as S_u.
- S(1) = dS(0). When the system value decreases, S(1) is also denoted as S_d.

$$f_u = \max(0, uS(0)-C_{xm}) \qquad (5)$$
$$f_d = \max(0, dS(0)-C_{xm}) \qquad (6)$$

Finally, notice one more thing before we apply the model: $v_{IP}(t)$ does not depend on which pattern is being employed. The market does not “see” the pattern; it only sees that the system provides some number of information sources. The market also “sees” availability, which does depend on the pattern being used.

Generalization of the two-step binomial valuation follows the same principles, solving the tree by calculating the values from the end nodes to the beginning. We also enforce two other conditions in order to follow this formulation of binomial valuation consistently:

$$dS(0) < C_{xm} < uS(0), \text{ and} \qquad (7)$$
$$d < 1 + r < u \qquad (8)$$

The condition enforced in $d < 1 + r < u$ suggests that the coefficient for an unfavorable outcome should be less than the interest rate plus one, whereas the system coefficient for a favorable outcome should be greater than this [Shreve 2005].

For each design choice, namely the current Client-Server system, a change to a Client-Dispatcher-Server, a Proxy, or a Broker design means that there is the opportunity to add multiple information resources at multiple times. That is equivalent to exercising the option multiple times [Meinshausen 2004]. We calculate a multiple-exercise option by calculating the option value for one-, two-, and three-period exercise strategies independently and *adding* the resulting option values.

Based on our cost assumptions, as shown in Table 4, the Proxy and Dispatcher design have the same switching and maintenance costs; therefore we model them with one tree. We assume V_S, introduced in Equations (1) and (4), to be a base value of \$2750. For simplicity, we assume that this value does not change over the timeline of the options we are evaluating. Also, since it represents the same part of the system in each of the designs, this value does not change across Client-Server, Client-Server-Dispatcher, Proxy, or Broker designs either. The utility values v_{IP} and v_A are obtained from Figure 3 and Figure 5.

We will now consider two scenarios and compare their option values. First, we consider which design choice would provide the most value if only modifiability of new information sources was of concern. We present these calculations in Section 3.6.1. This first scenario assumes that availability will not be a critical attribute to manage for the company. In our calculations therefore, we did not assume any added system utility value occurring as a result of the impact of availability to the users of this system. However, patterns carry within themselves different vulnerabilities and strengths for availability, as well as other quality attributes.

In the case that availability turns out to be critical and the company recognizes the need to plan for this, the option valuation needs to take into consideration the associated costs and utilities associated with the patterns regarding how they respond to availability. In this scenario, we do the calculations in a similar way. However, due to the availability vulnerability of the Broker design, we do not add any utility benefit from availability, which we present in Section 3.6.2.

3.6.1 Option Value: Modifiability

First, we consider modifiability to be the only quality attribute of concern in making the pattern choices. Figure 7 shows our calculations. The underlined values are the favorable and unfavorable outcome coefficients, *u* and *d*, that we calculate from the given system values. The p values are calculated based on Equation (3). The end nodes of each tree are calculated using Equations (5) or (6). The upper cells represent the system value, generated using formula (1). The shaded cells are the value of the option, based on (2).

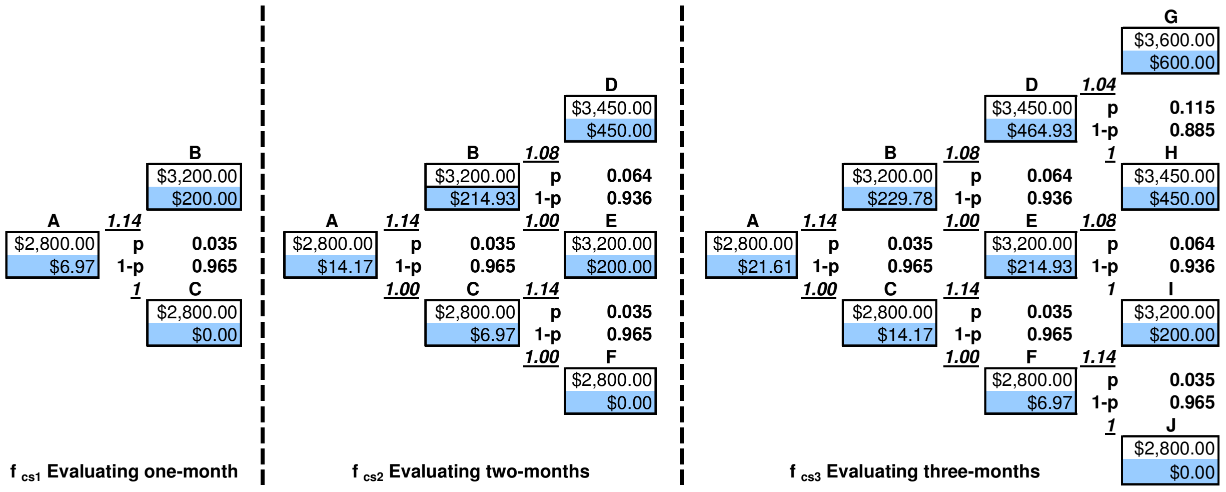


Figure 7: Evaluation of the Options in Client-Server Pattern with New Information Resources

To see in detail how these values are computed, consider cell D in the evaluation of the two-month case in Figure 7. The upper value in this cell is calculated as

$$S_{uu} = V_S + V_{IPcs}(2) = 2750 + 700 = 3450$$

The lower value is calculated as

$$f_{uu} = \max(0, S_{uu} - C_{csm})$$

The option value of the Client-Server system, f_{cs}, is then calculated as

$$f_{cs} = f_{cs1} + f_{cs2} + f_{cs3}$$

$$f_{cs} = 6.97 + 14.17 + 21.61 = 42.75 \qquad (9)$$

The same logic is used in calculating the option values of the Client-Dispatcher-Server, Proxy, and Broker designs. We use the matching maintenance costs for each design that we presented in Table 4. Figure 8 shows the corresponding valuation for Client-Dispatcher-Server or Proxy.

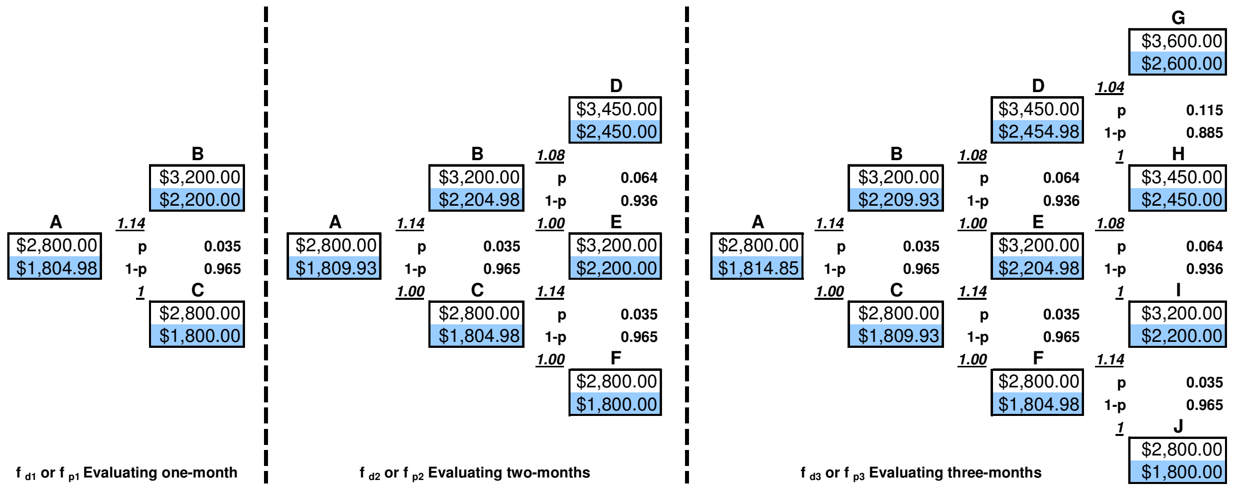


Figure 8: Evaluation of the Options in Dispatcher or Proxy Patterns with New Information Resources

The option value of the Client-Dispatcher-Server or Proxy patterns in the system, f_d or f_p, calculates to

$$f_p = f_d = f_{d1} + f_{d2} + f_{d3}$$

$$f_p = f_d = 1804.98 + 1809.93 + 1814.85 = 5429.75$$

Lastly, looking at the system with the Broker pattern evaluates to the option values as shown in Figure 9.

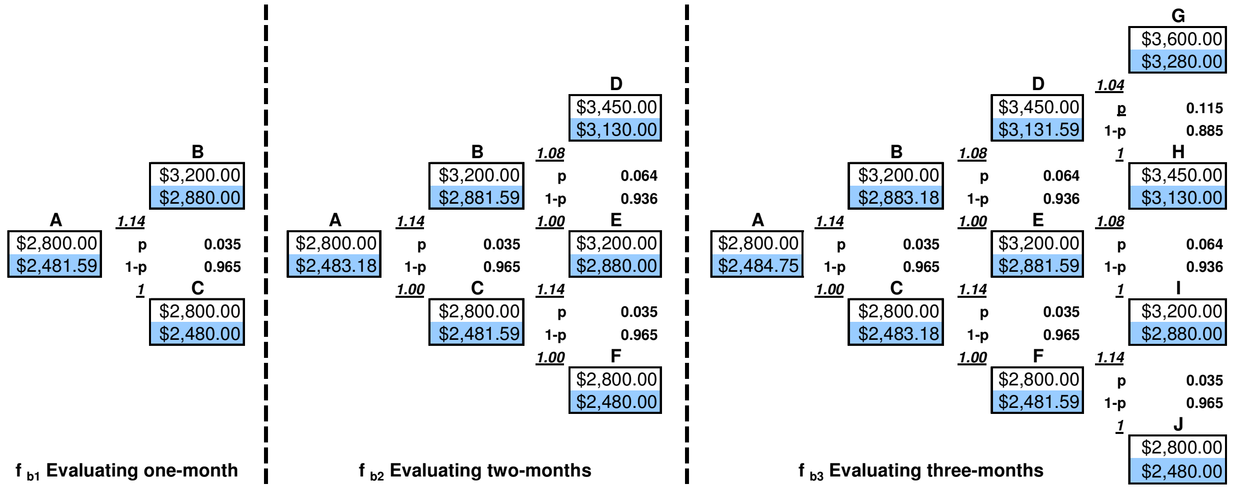


Figure 9: Evaluation of the Options in Broker Pattern with New Information Resources

The option value when using the Broker pattern is

$$f_b = f_{b1} + f_{b2} + f_{b3}$$

$$f_b = 2481.59 + 2483.18 + 2484.75 = 7449.52$$

Clearly, the value the Broker pattern brings to the system is higher than all the other possible patterns. When the switching costs (presented in Table 4) are taken into consideration, the Broker provides better value. Spending even \$4500 to switch to a Broker design creates more value (7449.52 – 4500 = 2949.52), than switching to a Client-Dispatcher-Server or Proxy (5429.75 – 4100 = 1329.75).

3.6.2 Option Value: Modifiability and Availability

Next, we take into consideration not only the utility value the patterns have as a result of modifiability but also the utility they do or do not exploit due to availability. The reasoning we follow is the same as with the calculations we demonstrated in Section 3.6.1. However, in considering the patterns, we also take into consideration the utility and maintenance costs due to availability. The system values of the different alternatives would be generated using $S = V_s + v_{IP} + v_A$ for the Client-Dispatcher-Server and Proxy designs. The availability utility to be gained from the current Client-Server and Broker designs are below the desired quality-attribute-response level expected: hence, they do not provide added benefit. This is represented by not considering the availability utility value as $S = V_S + v_{IP}$. An architectural pattern with vulnerability in respect to a certain quality attribute may still bring some utility value or may actually cause the overall design to lose value. Here, we make a simplification and assume v_A as zero for these cases where the design choices have vulnerabilities with respect to availability.

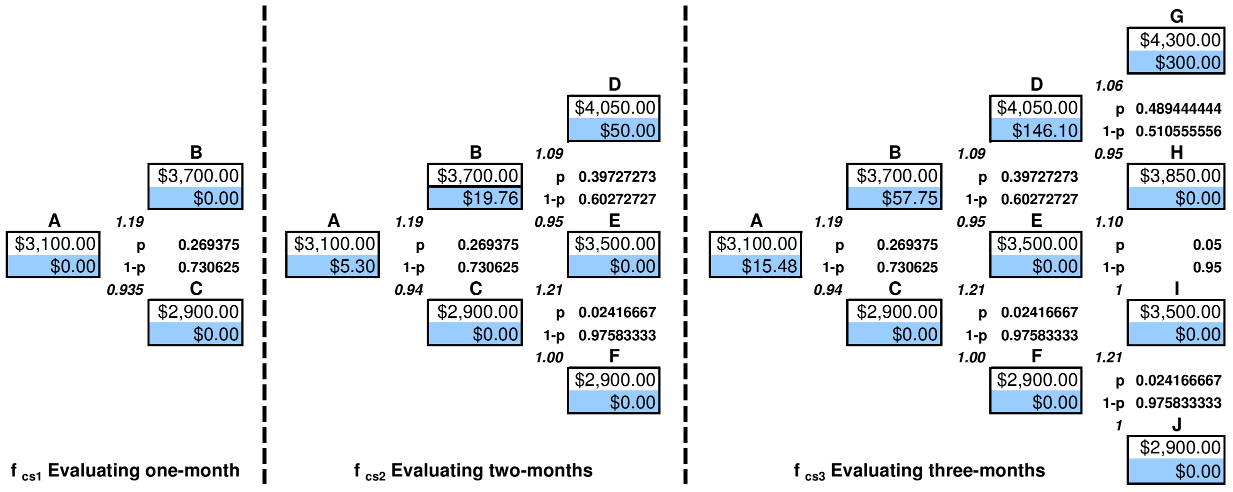


Figure 10: Evaluation of the Options in Client-Server Pattern with New Information Resources and Availability

As Figure 10 clearly shows, when availability becomes important, the current design is no longer feasible. The system gains no utility from availability; however, availability must be maintained for the system as part of the exercise strategy.

The total option value of the Client-Server system then comes to \$20.78 when we sum f_{cs1}, f_{cs2}, and f_{cs3} from Figure 10. The Client-Dispatcher-Server and Proxy designs, on the other hand, demonstrate a much more feasible outlook, as shown in Figure 11.

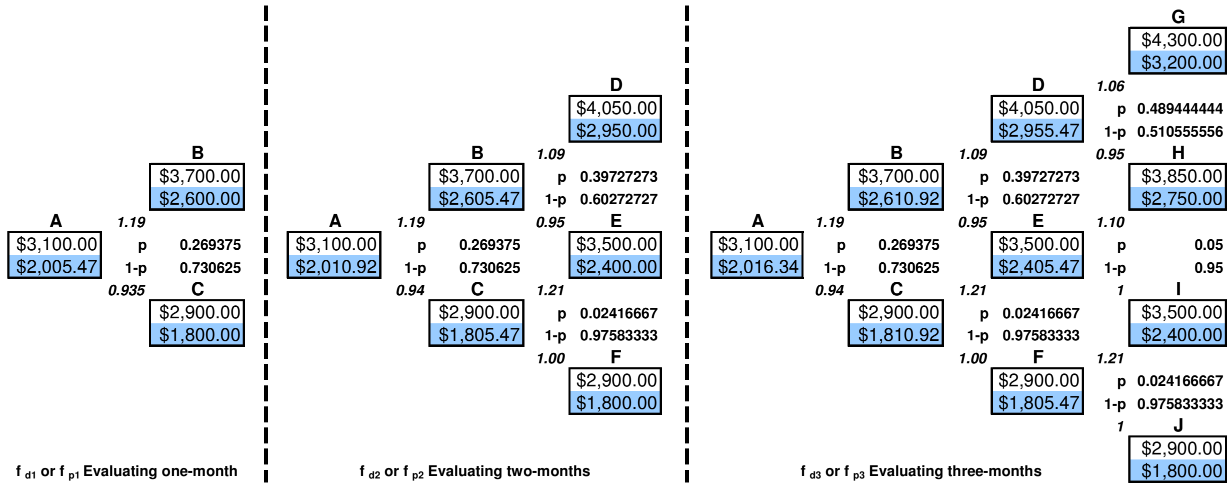


Figure 11: Evaluation of the Options in Client-Dispatcher-Server and Proxy Patterns with New Information Resources and Availability

The total option value of the Client-Dispatcher-Server or Proxy designs increases as a result of their strength in dealing with availability. This value evaluates as

$$f_p = f_d = f_{d1} + f_{d2} + f_{d3}$$
$$f_p = f_d = 2005.47 + 2010.92 + 2016.34 = 6032.73$$

This increase is due in part to the low cost of the availability maintenance in comparison to the utility obtained from it. For example, based on the cost assumptions of Table 4 and Figure 4, cell G in the tree of Figure 11 (where the evaluation is done for the three-month period) would evaluate as

$$f_{uuu} = \max(0, S_{uuu} - C_{dm}) = \max(0, 2750 + 850 + 700 - (1000 + 100))$$
$$= \max(0, 4300 - 1100) = 3200$$

Lastly, we calculate the value generated with the Broker design in the scenario that availability becomes a critical attribute in the market. Figure 12 shows these calculations. Despite the high advantage the Broker design holds for modifiability of adding information resources, due to its availability vulnerability, its option value drops.

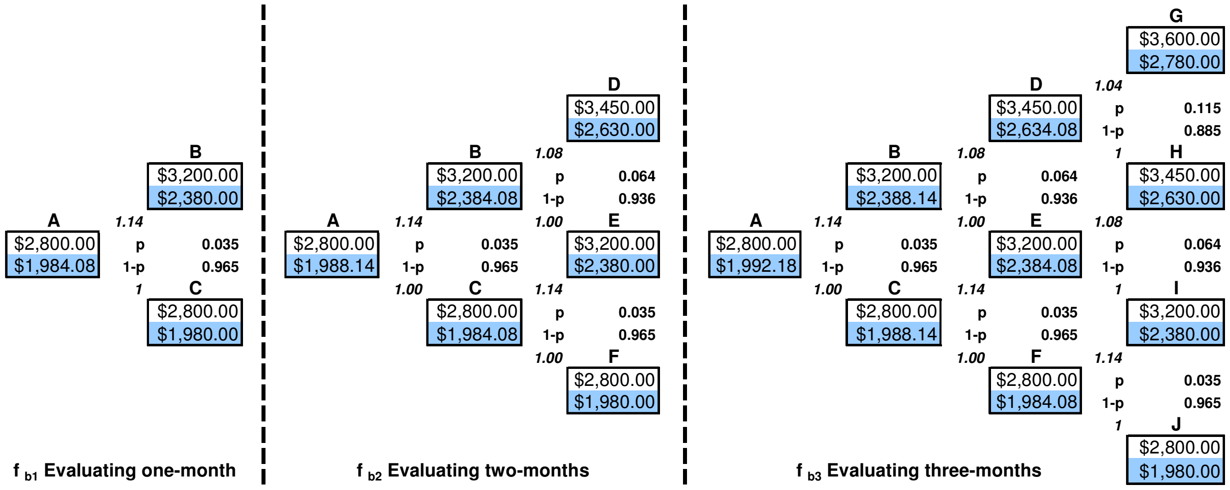


Figure 12: Evaluation of the Options in Broker Pattern with New Information Resources and Availability

The total option value in this case is

$$f_b = f_{b1} + f_{b2} + f_{b3}$$

$$f_b = 1984.08 + 1988.14 + 1992.18 = 5964.40$$

In this scenario, the company is better off *not* changing to a Broker design because the Client-Dispatcher-Server or Proxy choices can better accommodate the importance of availability in the market. The values of those choices are higher than that of the broker, despite its advantage in modifiability. This conclusion also holds when the switching costs (presented in Table 4) are taken into consideration. Spending $4500 to switch to a Broker design creates value (5964.40–4500 = 1464.40), but switching to Client-Dispatcher-Server or Proxy is *more* advantageous (6032.73 – 4100 = 1932.73).

3.7 DISCUSSION

We originally set out to answer two questions: "How and when should BizCo change the current architecture in response to new availability and information source demands?" and "Which potential design path has a higher option value in evolving the architecture?"

Our calculations show that BizCo should *wait* to make a change irrespective of the pattern employed because an early exercise does not provide a higher payoff than the value of the option in any of the nodes we calculated in Figure 8, Figure 9, Figure 11, and Figure 12. There is a value to postponing such decisions! Further, the option value of the patterns is dependent on not just the changes in market conditions but also the quality attributes in which BizCo is interested. If only modifiability is critical, the Broker pattern is the clear winner. However, to obtain options in availability, BizCo should pick the Proxy or Client-Dispatcher-Server pattern.

Value-based comparisons between those two patterns can be conducted where performance is evaluated as well, similar to what we have demonstrated with modifiability and availability attributes. Although we have not shown the calculations for performance in this report, they are a straightforward extension of the model presented.

The example we used is intentionally oversimplified: this was done to focus on demonstrating our approach and reasoning. The important points of our approach are

1. These conclusions were drawn by making a combined economic and technical analysis of each architectural pattern option considering possible future market scenarios and multiple quality attribute dimensions.
2. This technique is practical because the information required to make this analysis can be readily elicited from project stakeholders.

The implication of our approach is that there is close coordination and collaboration between the technical and business experts in the organization. Incorporation of economic reasoning into making software architecture decisions by focusing on the utility to be gained from quality attribute requirements, uncertainty, and business goals—as our approach demonstrates—provides a technique where the language of both technical and business experts can be used. We have seen this as a side benefit of other economics-driven approaches to architectural analysis as well [Moore 2003].

4 Related Work

Two research directions emerge from an examination of the prior work in the domain of applying real options theory to the valuation of architectural decisions:

1. work that focuses on the validation and application of the seminal DSM and corresponding NOV model of Baldwin and Clark in the context of object- and aspect-oriented modular designs
2. work that investigates the direct application of different kinds of real options valuation and problem types to software architecting

Baldwin and Clark argue that modularity in design creates design options in the sense that a modular design process creates several options as a result of the modules. Sullivan and colleagues, in validating this theory, suggest that architects are drawn to information-hiding interfaces to create the "right" but not the obligation to change the design in the future [Sullivan 2001]. The option value of each module then depends on the number of external dependencies for that module because the cost to replace a module increases if there are many other modules that are affected by the change. DSM and NOV theory has also been used in value-based analysis of modularity in aspect-oriented design. Sullivan and colleagues compare the value of modularity in aspect- and object-oriented designs over an example using NOV calculations as value indicators [Sullivan 2005]. Lopes and Bajracharya, similarly, look at the module dependencies and their added value in aspect-oriented designs [Lopes 2005]. Both works use DSMs for dependency analysis and tracking the design rules.

These studies often view architectural design decisions as a series of decisions about how to modularize the system. They do *not* consider the potential resulting structural impact of quality attributes on the architecture that may be better captured with patterns rather than modularity alone. Lopes and Bajracharya, in particular, look mostly at the functional decomposition aspect of a modular structure in its real options analysis and do not take quality attributes into consideration in making such architectural restructuring decisions [Lopes 2005].

Our approach to valuing architectural patterns and making decisions about how to evolve an existing architecture based on their value clearly shows that the expected utility from quality attribute scenarios determines the value of design. The example we demonstrated contributes to the understanding of how patterns may behave in design. Patterns, both at the detailed design level using the "Gang-of-Four" patterns [Gamma 1995] and at an architecting level using architectural patterns [Buschmann 1996] or styles [Shaw 1996] have attracted ample attention. Cai recognizes how the application of one pattern may have a pervasive impact on the rest of the design by creating new constraints and modeling them using DSMs [Cai 2006]. However, some of these constraints can be observed only at the time of implementation, which may be too late for planning around them. Quality attribute scenarios as a variable of valuation may assist the elicitation of these constraints earlier, when architectural commitments are being made.

An appealing aspect of the application of options theories, particularly real options analysis, is in characterizing software development projects in terms of different types of real options. Why and when does it make sense to defer making design decisions, as in extreme programming [Beck 2000]? Erdogmus and Favaro look at Extreme Programming in terms of how deferring design decisions creates options because, as more information about requirements becomes available, there is less uncertainty [Erdogmus 2002b]. Bahsoon and Emmerich assess refactoring value using the Black-Scholes-Merton formula, making the assumption that there is one expiration time to exercise the refactoring decision in design [Bahsoon 2003]. Chalasani, Jha, and Sullivan define prototyping activities as creating options on design [Chalasani 1997]. Favaro defines an options-based approach to investing in reuse structures [Favaro 1998]. Cai, Manvendra, and Sullivan explore the value of dynamic switching based on binomial tree valuation [Cai 2002]. Asundi and Kazman discuss how the portfolio of options approach can be incorporated into the dependency structure of architectural strategies elicited in applying the CBAM to architecture evaluation [Asundi 2001a].

These studies contribute to the better understanding of the applicability of options theory in general terms to software architecting—identifying how different types of options can be defined depending on project characteristics. Waiting to invest, deferring decisions, project growth, flexibility, exiting an opportunity, and learning are some types of real options that can be embedded in software projects.

5 Conclusions and Future Directions

In this report, we introduced an economics-driven approach for valuing patterns based on quality attributes. Our approach focuses on patterns and quality attributes and creates a common vocabulary for the discussion and evaluation of software architecture decisions, shared between architects and managers. Modularity becomes an appealing design strategy when there is an uncertainty of change and a need to evolve an existing architecture. Our example shows that modularity should not be the only criterion used to make sound architectural decisions; addressing other quality attributes such as performance, availability, or usability is also very important. Our point is that such “additional” considerations create options in projects [Wang 2006], but these options are seldom quantified, particularly in the face of uncertain future market conditions.

Previous work in applying real options analysis to valuing modular designs has generated several important observations about the value of a module. Baldwin and Clark observed that not all modules have equal value; those with higher technical risk and complexity that offer avenues of independent exploration are the most valuable ones [Baldwin 2000]. Sullivan and colleagues observed that environment determines a module’s option value; if there is little value to be gained by replacing a module, no matter how complex it is, then it has low technical potential—hence a lower value [Sullivan 2001].

Our approach thus generalizes and extends previous research by simultaneously considering multiple quality attributes and their future potential utility. Quality attributes derive primarily from a system’s software architecture, and consequently quality attribute requirements are a driving force for architectural design [Bass 2003]. This report recognizes that architectural patterns create quality attribute options that have value and shows how the process of designing architectures can explicitly take this value creation into account in a manageable way. Our example shows that formulating an architecture evolution problem by taking into consideration future uncertainties requires the following:

1. identification of business goals
2. identification of uncertainty that the business needs to manage
3. identification of quality attribute scenarios
4. identification of possible real options to embed
5. identification of applicable tactics/patterns
6. comparative evaluation, and finally
7. generation of a strategy by highlighting key metrics to observe, along with their corresponding patterns and tactics, as future courses of action

Bringing economic reasoning to making software architecture decisions as demonstrated by our approach in this report has several managerial implications. Business experts could use such techniques for taking advantage of technical insights when making product decisions. Similarly, these technical insights can enable business experts to manage uncertainty in the market better and be better prepared to respond to emerging conditions. A desirable outcome would be for the business experts to be informed of and manage the cost/benefit implications of technical decisions made at an architectural level. Similar benefits exist for software architects. They can use such techniques as tools to demonstrate the positive and negative potential of architectural decisions at the business level. They can prepare their architectures better for market uncertainty and evolution and therefore better manage their project quality and schedule goals.

Our ultimate goal is to develop cost/benefit heuristics for architecture design and evolution that can be employed by an architect or project management team. In this regard, we plan several directions for future work. First, we need to understand the types of data architects can readily collect as they design, which can assist in filling the gaps of existing valuation models. Second, we need to be able to give guidance to practitioners regarding when and how they can recognize uncertainty variables in quality attributes and plan for them. Third, we will explore whether it is possible to categorize patterns based on the options they create in design and understand how to effectively create portfolios of options.

Bibliography

URLs are valid as of the publication date of this document.

[Amram 1999]
Amram, M. & Kulatilaka, N. *Real Options: Managing Strategic Investment in an Uncertain World*. Cambridge, Massachusetts: Harvard Business School Press, 1999.

[Asundi 2001a]
Asundi, J. & Kazman R. *A Foundation for the Economic Analysis of Software Architectures*. http://www.cs.virginia.edu/~sullivan/edser3/asundi.pdf (2001).

[Asundi 2001b]
Asundi, J.; Kazman R.; & Klein, M. *Using Economic Considerations to Choose Among Architecture Design Alternatives* (CMU/SEI-2001-TR-035, ADA3399151). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
http://www.sei.cmu.edu/publications/documents/01.reports/01tr035.html.

[Bahsoon 2003]
Bahsoon, R. & Emmerich, W. *ArchOptions: A Real Options-Based Model for Predicting the Stability of Software Architectures*.
http://www.cs.ucl.ac.uk/staff/w.emmerich/publications/ICSE2003/RealOptions /BahsoonEmmerich.pdf (2003).

[Baldwin 2000]
Baldwin, C. Y. & Clark, K. B. *Design Rules: Volume 1, The Power of Modularity*. Cambridge, MA: MIT Press, 2000.

[Baldwin 2006]
Baldwin, C. Y. & Clark K. B. “Between ‘Knowledge’ and ‘the Economy:’ Notes on the Scientific Study of Designs,” 299-328. *Advancing Knowledge and the Knowledge Economy*, edited by Forey, D. & Kahin, B. Cambridge, MA: MIT Press, 2006.

[Bass 2003]
Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice, Second Edition.* Boston, MA: Addison Wesley, 2003.

[Beck 2000]
Beck, K. *Extreme Programming Explained*. Boston, MA: Addison-Wesley, 2000.

[Black 1972]
Black, F. & Scholes, M. “The Valuation of Option Contracts and a Test of Market Efficiency.” *Journal of Finance 27*, 1 (May 1972): 399-418.

[Boehm 1981]
Boehm, B. W. *Software Engineering Economics*. Upper Saddle River, NJ: Prentice Hall, 1981.

[Boehm 2000]
Boehm, B. & Sullivan, K. J. “Software Economics: A Roadmap,” 319-343. *Proceedings of the Conference on the Future of Software Engineering*. Limerick, Ireland, June 4-11, 2000. New York, NY: ACM Press, 2000.

[Buschmann 1996]
Buschmann, F.; Meunier R.; Rohnert, H.; Sommerlad, P.; & Stal, M. *Pattern-Oriented Software Architecture A System of Patterns.* New York, NY: John Wiley & Sons, 1996.

[Cai 2002]
Cai, Y.; Manvendra, P. S.; Sorokin, P.; & Sullivan, K. *Stochastic Optimal Switching: Theory, Architecture, Prototype*. http://www.cs.drexel.edu/~yfcai/Papers/edser2002.PDF (2002).

[Cai 2006]
Cai, Y. “Modularity in Design: Formal Modeling and Automated Analysis.” PhD diss., School of Engineering and Applied Science, University of Virginia, 2006.

[Chalasani 1997]
Chalasani, P.; Jha, S.; & Sullivan, K. *An Options Approach to Software Prototyping* (CMU-CS-97-161). Pittsburgh, PA: School of Computer Science, Carnegie Mellon University, 1997. http://reports-archive.adm.cs.cmu.edu/anon/1997/abstracts/97-161.html.

[Clements 2002]
Clements, P.; Kazman, R.; & Klein, M. *Evaluating Software Architectures: Methods and Case Studies*. Boston, MA: Addison-Wesley, 2002.

[Cox 1979]
Cox, J.; Ross, S.; & Rubinstein, M. “Option Pricing: A Simplified Approach.” *Journal of Financial Economics 7*, 3 (September 1979): 229-263.

[Dobrica 2002]
Dobrica, L. F. & Niemela, E. “A Survey on Software Architecture Analysis Methods.” *IEEE Transactions on Software Engineering 28*, 7 (July 2002): 638-653.

[Erdogmus 2002a]
Erdogmus, H. Ch. 43, “Valuation of Learning Options in Software Development Under Private and Market Risk,” 503-552. *The Engineering Economist.* Boston, MA: Addison-Wesley, 2002.

[Erdogmus 2002b]
Erdogmus, H. & Favaro, J. “Keep Your Options Open: Extreme Programming and the Economics of Flexibility,” 503-552. *Extreme Programming Perspectives*. Boston, MA: Addison Wesley, 2002.

[Favaro 1998]
Favaro, J. M.; Favaro, K. R.; & Favaro, P. F. “Value Based Software Reuse Investment.” *Annals of Software Engineering 5,* 0 (January 1998): 5-52.

[Gamma 1995]
Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.; Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[Golden 2005]
Golden, E.; John, B. E.; & Bass, L. “The Value of a Usability-Supporting Architectural Pattern in Software Architecture Design: A Controlled Experiment,” 460-469. *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*. Saint Louis, MO, May 15-21, 2005. Washington, DC: IEEE Computer Society, 2005.

[Hannemann 2002]
Hannemann, J. & Kiczales, G. “Design Pattern Implementation in Java and AspectJ,” 161-173. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 2002).* Seattle, WA, November 4-8, 2002. New York, NY: Association for Computing Machinery, 2002.

[Hull 2006]
Hull, J. C. *Options, Futures, and Other Derivatives, Sixth Edition*. Upper Saddle River, NJ: Prentice Hall, 2006.

[Lopes 2005]
Lopes, C. V. & Bajracharya, S. “An Analysis of Modularity in Aspect-Oriented Design.” *Proceedings of the 4th International Conference on Aspect-Oriented Software Development (AOSD 2005).* Chicago, IL, March 14-18, 2005. New York, NY: Association for Computing Machinery, 2005.

[Mauboussin 1999]
Mauboussin, M. J. *Get Real: Using Real Options in Security Analysis.* http://www.capatcolumbia.com/Articles/FoFinance/Fof10.pdf (1999).

[Meinshausen 2004]
Meinshausen, N. & Hambly, B. M. “Monte Carlo Methods for the Valuation of Multiple-Exercise Options.” *Mathematical Finance 14*, 4 (October 2004): 557-583.

[Merton 1998]
Merton, R. C. “Applications of Option-Pricing Theory: Twenty-Five Years Later.” *The American Economic Review 88,* 3 (1998): 323-349.

[Moore 2003]
Moore, M.; Kazman, R.; Klein, M.; & Asundi, J. “Quantifying the Value of Architecture Design Decisions: Lessons from the Field,” 557-562. *Proceedings of the 25th International Conference on Software Engineering.* Portland, OR, May 3-10, 2003. Los Alamitos, CA: IEEE Computer Society, 2003.

[Shaw 1996]
Shaw, M. & Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline.* Upper Saddle River, NJ: Prentice Hall, 1996.

[Shaw 2005]
Shaw, M.; Arora, A.; Butler, S.; Poladian, V.; & Scaffidi, C. *In Search of a Unified Theory for Early Predictive Design Evaluation for Software* (Technical Report CMU-ISRI-05-114). Pittsburgh, PA: Carnegie Mellon University Institute for Software Research International, 2005.

[Shreve 2005]
Shreve, S. E. *Stochastic Calculus for Finance I: The Binomial Asset Pricing Model.* New York, NY: Springer, 2005.

[Sullivan 1999]
Sullivan, K. J.; Chalasani, P.; Jha, S.; & Sazawal, V. Ch. 10, "Software Design as an Investment Activity: A Real Options Perspective," 215-262. *Real Options and Business Strategy: Applications to Decision Making.* London, UK: Risk Books, 1999.
http://www.cs.virginia.edu/~sullivan/publications/RiskBooksChapter.pdf.

[Sullivan 2001]
Sullivan, K. J.; Griswold, W.; Cai, Y.; & Hallen, B. "The Structure and Value of Modularity in Software Design," 99-108. *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering.* Vienna, Austria, September 10-14, 2001. New York, NY: Association for Computing Machinery, 2001.

[Sullivan 2005]
Sullivan K. J.; Griswold, W. G.; Song, Y.; Cai, Y.; Shonle, M.; Tewari, N.; & Rajan, H. "Information Hiding Interfaces for Aspect-Oriented Design," 166-175. *Proceedings of the Joint 10th European Software Engineering Conference and 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering.* Lisbon, Portugal, September 5-9, 2005. New York, NY: Association for Computing Machinery, 2005.

[Wang 2006]
Wang, T. & de Neufville, R. *Identification of Real Options "in" Projects.*
http://ardent.mit.edu/real_options/Real_opts_papers/Identification%20of%20Real%20Option%20in%20Projects%20INCOSE.pdf (2006).

[Wojcik 2006]
Wojcik, R.; Bachmann, F.; Bass, L.; Clements, P.; Merson, P.; Nord, R.; & Wood, B. *Attribute-Driven Design (ADD), Version 2.0* (CMU/SEI-2006-TR-023). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006.
http://www.sei.cmu.edu/publications/documents/06.reports/06tr023.html.

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*

1. **AGENCY USE ONLY** (Leave Blank)	2. **REPORT DATE** May 2007	3. **REPORT TYPE AND DATES COVERED** Final
4. **TITLE AND SUBTITLE** Quality-Attribute-Based Economic Valuation of Architectural Patterns		5. **FUNDING NUMBERS** FA8721-05-C-0003
6. **AUTHOR(S)** Ipek Ozkaya, Rick Kazman, & Mark Klein		
7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213		8. **PERFORMING ORGANIZATION REPORT NUMBER** CMU/SEI-2007-TR-003
9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** HQ ESC/XPK, 5 Eglin Street, Hanscom AFB, MA 01731-2116		10. **SPONSORING/MONITORING AGENCY REPORT NUMBER** ESC-TR-2007-003
11. **SUPPLEMENTARY NOTES**		
12A **DISTRIBUTION/AVAILABILITY STATEMENT** Unclassified/Unlimited, DTIC, NTIS		12B **DISTRIBUTION CODE**

13. **ABSTRACT (MAXIMUM 200 WORDS)**

Quality attribute requirements are a driving force for software and system architecture design. Architectural patterns can be used to achieve quality attribute requirements. Consequently architectural patterns generate value based on the present and future *utility* of the quality attributes they achieve. This report makes the case that architectural patterns carry economic value in part in the form of real options, providing software architects the right, but not the obligation, to take subsequent design actions. The report shows, via a simple example, how an analysis of the options embodied within architectural patterns allows an architect or manager to make reasoned choices about the future value of design decisions, considering this value along multiple quality attribute dimensions.

14. **SUBJECT TERMS** economics-driven architecting, quality attributes, real options, financial options, value-driven software engineering			15. **NUMBER OF PAGES** 50
16. **PRICE CODE**			
17. **SECURITY CLASSIFICATION OF REPORT** Unclassified	18. **SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	19. **SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	20. **LIMITATION OF ABSTRACT** UL